AD-A120 204 MARYLAND UNIV COLLEGE PARK DEPT OF COMPUTER SCIENCE F/G 9/2
FORMAL SPECIFICATION AND VERIFICATION OF DISTRIBUTED SYSTEMS.(U)
JUN 82 B CHEN, R T YEH F49620-80-C-0001
UNCLASSIFIED AFOSR-TR-82-0863 NL

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(2)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 82-0863	2. GOVT ACCESSION NO. AD-A120204	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) FORMAL SPECIFICATION AND VERIFICATION OF DISTRIBUTED SYSTEMS		5. TYPE OF REPORT & PERIOD COVERED TECHNICAL
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Bo-Shoe Chen and Raymond T. Yeh		8. CONTRACT OR GRANT NUMBER(s) F49620-80-C-0001
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Computer Science University of Maryland College Park MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS PE61102F; 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Directorate of Mathematical & Information Sciences Air Force Office of Scientific Research Bolling AFB DC 20332		12. REPORT DATE June 1982
		13. NUMBER OF PAGES 13
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

AD A120204

DTIC FILE COPY

DTIC
ELECTE
OCT 13 1982
E

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
The authors develop an event-based model to specify formally the behavior (the external view) and the structure (the internal view) of distributed systems. Both control-related and data-related properties of distributed systems are specified using two fundamental relationships among events; the 'happens before' realtion, representing time order; and the 'enabling' relation, representing causality. No assumption about the existence of a global clock is made in the specifications.

(CONTINUED)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ITEM #20, CONTINUED:

The correctness of a design can be proved before implementation by checking the consistency between the behavior specification and the structure specification of a system. Important properties of concurrent systems such as 'mutual exclusion', 'concurrency', and other 'safety' and 'liveness' properties can be specified and verified.

FORMAL SPECIFICATION AND VERIFICATION OF DISTRIBUTED SYSTEMS

by Bo-Shoe Chen* and Raymond T. Yeh

Department of Computer Science
University of Maryland
College Park, Maryland 20742

Abstract

We develop an event-based model to specify formally the behavior (the external view) and the structure (the internal view) of distributed systems. Both control-related and data-related properties of distributed systems are specified using two fundamental relationships among events; the "happens before" relation, representing time order; and the "enabling" relation, representing causality. No assumption about the existence of a global clock is made in the specifications.

The correctness of a design can be proved before implementation by checking the consistency between the behavior specification and the structure specification of a system. Important properties of concurrent systems such as "mutual exclusion," "concurrency," and other "safety" and "liveness" properties can be specified and verified.

1. Introduction

Computations of distributed systems are extremely difficult to specify and verify using traditional techniques because the systems are inherently concurrent, asynchronous and nondeterministic. Furthermore, computing nodes in a distributed system may be highly independent of each other, and the entire system may lack an accurate global clock.

In this paper, we develop an event-based model to specify formally the behavior (the external view) and the structure (the internal view) of distributed systems. Both control-related and data-related properties of distributed systems are specified using two fundamental relationships among events; the "happens before" relation, representing time order; and the "enabling" relation, representing causality. No assumption about the existence of a global clock is made in the specifications.

The correctness of a design can be proved before implementation by checking the consistency between the behavior specification and structure specification of a system. Important properties of concurrent systems such as "mutual exclusion," "concurrency," and other "safety" and "liveness" properties can be specified and verified.

Moreover, since the specification technique defines the orthogonal properties of a system separately, each of them can then be verified independently. Thus, the proof technique avoids the exponential state-explosion problem found in state-machine specification techniques.

2. Conceptual Modelling

A distributed system may be described from two different points of view. From a designer's viewpoint, it consists of local processes interacting with users and communicating among themselves via the service of communication medium. Each local process can be described by the operations responding to user's commands, messages from other processes or internal clocks. The structure is depicted in Figure 1.

From a user's viewpoint, a distributed system is a black box, or a shared server with only the interfaces visible to him, as shown in Figure 2. In this case, except for performance issues, there is no difference in functionality between a distributed system and a centralized one. The only things interesting are what kind of messages or events may happen in the interfaces and what are the relationships among the messages or the events. We call such kind of interface description of a system, its behavior specification.

3. The Event Model

We consider the behavior of a system to be a set of computation histories characterized by "events." The model in which our specification is based upon, therefore, consists of events and their relationships.

3.1 Event

An event is an instantaneous, atomic state transition in the computation history of a system. Examples of events are the sending, the receiving, and the processing of messages. By "instantaneous" we mean an event takes zero-time to happen. By "atomic" we mean an event happens completely or not at all.

3.2 Event Relationships

* Dr. Chen is now working in Bell Laboratories, IH 6C-309, Naperville, IL 60566

3.2.1 Time Ordering: '->'

In describing the time ordering among events, a system-wide reliable clock is usually assumed to order totally the events in a centralized system. Unfortunately, the assumption of the existence of such a global clock is too strong in describing the computation of a distributed system which is inherently concurrent, asynchronous and nondeterministic. Theoretically speaking, in some extreme case, it is impossible to order two events totally when they happen in two geographically separated places. Practically speaking, implementing such a global clock is quite expensive and unnecessary in a distributed system having highly autonomous computing nodes. We give up the global clock assumption and come out with a partial ordering relation-the "preceding" relation denoted by '->', to represent the time concept [GRE77, LAM78].

The interpretation of '->' as a time ordering means that, if e1 and e2 are events in a system and e1->e2, then e1 happens before e2 by any measure of time. To understand the meaning of '->', let us look into Figure 3. Each vertical line in Figure 3 represents the computation history of a (sequential) "process." By a "process" we mean, an autonomous computing node having its own "local" clock; different processes may use different time scales. The dots denote events and the dotted line between events denote messages. The relation '->' has the following properties:

(1) If e1 and e2 are events in the same process, and e1 comes before e2, then e1->e2 (e.g. p1->p2 in Figure 3);

(2) If e1 is the sending event of a message by one process and e2 is the receiving event of the message by another process, then by the law of 'causality', e1->e2 (e.g. p1->q2 in Figure 3);

(3) (Transitivity property) If e1->e2 and e2->e3 then e1->e3 (e.g. p1->q3 in Figure 3);

(4) (Irreflexivity property) For every event e, ~(e->e);

(5) (Antisymmetry property) If e1->e2 then ~(e2->e1)

3.2.2 Concurrency

Two distinct events, say e1 and e2, are concurrent if ~(e1->e2) and ~(e2->e1) and denoted by e1//e2. In Figure 3, for example, although p1->q2 and q1->p2, there is no way to tell whether p1 or q1 comes first; they may be concurrent.

3.2.3 Enabling Relation =>

An important class of properties in communication systems is the guaranteed service of message transmission. Such kind of properties can be specified by the introduction of the enabling relation, denoted by "=>" between events. Two events, say a and b, satisfy the relation a=>b if the existence of event a will cause the occurrence of event b in the future. The relation => has the following properties:

(1) Events are enabled in the future,
if a => b then a -> b

(2) Anti-symmetry property,
if a => b then ~(b => a)

(3) Irreflexivity property,
~(a => a)

(4) Transitivity property,
if a => b and b => c then a => c

Properties (2) and (3) can be derived from (1) and the properties of relation -> , while (1) and (4) are essential axioms for the relation => .

3.2.4 System, Environment, Their Interfaces and Events

The event space in the computation history is categorized into three distinct domains: the system, the environment and the interfacial ports.

A system interacts with its environment by exchanging messages through unidirectional interfaces called ports, as depicted in Figure 4. An inport (outport) directs messages from the environment (system) to the system (environment).

Every port defines sequences of interfacial events. Every event in a port history is uniquely identified by an integer number, called ordinal number. Thus, a port history is a total ordering of events, although the events in system or in environment are only a partial ordering.

4. The Language EBS

Based on the concepts above, we developed a language called EBS (Event Based Specification Language) to specify the behavior of distributed systems. Instead of presenting the formal syntax of the language, we use examples to show up its expressive power.

4.1 Example 1: Reliable Transmission Systems

A reliable transmission system (RT) is one through which messages are transmitted without error, loss, duplication or reordering from an inport to an outport (see Figure 5). Although most physical communication media are unreliable that may lose, duplicate or reorder messages, almost all designers provide communication protocols (e.g. Alternate Bit Protocol) to convert them into logically reliable ones for the ease of application programs that build on the top of the communication systems.

The property that there is no loss of messages during the transmission means that every message sent from the inport A will eventually be transmitted to the outport B. This can be specified as follows:

```
(* RT11(A,B)  [1]  : No loss of messages*)
   ∀ a← A ∤ b← B
     a=> b;
```

Similarly, the property that messages at B are not generated internally or externally but are enabled by messages at A, is specified as follows:

```
(* RT12(A,B): no self-existing messages*)
   ∀ b← B ∤ a← A
     a=> b;

(* RT13(A,B): no internally or externally
              generated messages
*)
   ∀ b← B,  s← SYS, e← ENV      [2]
     (s =>b #> ∤ a← A a=>s=>b)  ^
     (e =>b #> ∤ a← A e=>a=>b)
```

The reserved word SYS (ENV) refers to the set of system (environment) events. The property that there is no duplication of messages is specified as follows:

```
(* RT14(A,B): no duplication of messages *)
   ∀ a← A, b1,b2← B
     a=>b1 ^ a=>b2 #> b1≠b2
```

which says that every sending event can only enable a unique receiving event. The property that the order of messages is preserved after transmission is specified as follows:

```
(* RT15(A,B): no out of order messages *)
   ∀ a1,a2← A, b1,b2← B
     a1=>b1 ^ a2=>b2
     #> (a1-> a2 ^ b1-> b2)  v
        (a1≠ a2 ^ b1≠ b2)  v
        (a2-> a1 ^ b2-> b1)
```

which says that if a1 is sent before a2 then it will also be received before a2. The property that the contents of messages are preserved after the transmission is specified as follows:

```
(* RT21(A,B): preservation of message
   contents*)
   ∀ a← A, b← B
     a=>b #> a.msg=b.msg
```

which says that the receiving and sending event carry the same message contents.

These are about the weakest properties that a reliable transmission system should have. A very good feature of this kind of orthogonal specification is that a specification can be easily adapted to different applications. For example, if we want to specify the behavior of a communication system which not only transmits messages reliably but also performs code conversions between computer systems communicating with each other using different codes (e.g., ASCII and EBCDIC), we need only change RT21 to

```
(# TR21(A,B): message transformer*)
   ∀ a←A, b← B
     a=> b #> b.msg= F(a.msg)
```

where F is the code conversion function, and leave others unchanged. This can also be seen from the specification of the following system.

4.2 Example 2: Unreliable Transmission System (UT)

An unreliable transmission system is the one through which messages may be lost, duplicated or reordered, but there is a non-zero probability of message transmission and no erroneous messages. Most physical communication media belong to this class.

The property that there is a non-zero probability of message transmission can be specified as

```
(*NZ(A,B): a nonzero probability of successful message transmission.
*)
  ∀  ai← A
     (∀ aj← A aj.msg= ai.msg
        #> ∤ ak← A aj-> ak ^ ak.msg= ai.msg)
     #> (∤ a← A, b← B
           a=> b^ a.msg= ai.msg ^ ai-> a)
```

which means that if a group of messages having the same contents are sent unboundedly then at least one of them will reach B.

The unreliable transmission system is specified as follows:

```
System  UT  (A : inport;
             B : outport);

Behavior

   (* A nonzero probability of successful
              message transmission.
   *)
      NZ(A,B);

   (* No self-existing messages *)
      RT12(A, B);
```

---

[1] We will use RT11 to name this property afterwards for convenience.

[2] The order of operator precedence in the language is as follows: (1) uniary operators; ∀ (for all), ∤ (there exists) and ~ (logical not); (2) relational operators: ← (belongs to), = (equivalent to), ≠ (equals to); (3) logical operators: v (logical or), ^ (logical and);(4) #> (logical implication).

```
(* No internally or externally generated
   messages *)
   RT13(A, B);

(* RT11, RT14 and RT15 are discarded
   which means that the system may
   lose, duplicate or reorder messages.
 *)

(*No erroneous messages *)
  RT21(A, B);

End behavior

End system.
```

5. Structure Specification and Verification

In a top-down hierarchical design, the service that a distributed system provides is described first by the behavior specification. Then the specification is decomposed, according to a design rationale, into a set of sub-systems communicating via the service of connection links. We call such kind of design (internal) description of a system, its structure specification. Once we get both behavior and structure specifications, the correctness of a design can be proved by checking the consistancy between these two specifications.

5.1 System Constructs

A subsystem is a building block of the whole system. The computation of a subsystem is described by a behavior specification, which can be further decomposed into a structure specification. In this way, our specification technique supports the hierarchical design methodology.

A link connects an outport of a subsystem to an inport of another subsystem. When two ports are linked, they are merged into a single port. The event semantics of a link are that ports are identical in the outport and the inport being linked together. By identical we mean two events are just the same; it is impossible to distinguish between them.

Note that a link is different from a reliable transmission system in that the latter introduces finite message delay as in a physical cable connection while the former transmits messages reliably and without any delay (i.e., instantaneously). Note also that two ports cannot be linked unless they have exactly the same message types.

5.2 Example 3: A Tandem Network

In packet-switched network, a packet of message, instead of sent directly from the source node to the destination node using a long-haul transmission line, is passed via some intermediate nodes between the source node and the destination node. A message is sent reliably from the source node to the intermediate node and then sent reliably from the intermediate node to the destination node. Thus, the structure of the communication system can be considered as consists of a set of reliable transmission sub-systems connecting in series, which, as a whole, provides the service of a reliable transmission system for the users of this packet- switched network. We call such a serial connection of two or more subsystems, a tandem (see Figure 6) network.

5.2.1 Verification of the Tandem Network

Since we are using the same mathematically sound notations (i.e., first-order logic and partial ordering relations), the verification process can be carried out as proving theorems.

Theorem 1. A tandem connection of two reliable systems behaves as a single reliable one.

Proof

The no loss property can be proved as follows:

(1) For all p in PA there is a q in PB such that p=> q (Since RT11 of SA)
(2) For all r in PC there is an s in PD such that r=> s (since RT11 of SB)
(3) Let q= r (since PB and PC are connected)
(4) p=> s (since => is transitive)

Other properties can be proved similarly, independent of one another.

Although the proofs of the theorems are carried out in a somehow informal way, they may actually be formalized. See [CHES2] for details of the verification.

5.3 Example 4: An Alternate-Bit Protocol

An Alternate-Bit Protocol is intended to provide a reliable message transfer over an unreliable transmission medium from a fixed sender or a fixed receiver. The service provided by this protocol is, thus, nothing more than that of a reliable transmission system.

The underlying communication medium is an unreliable one, which may lose, duplicate or reorder messages; however, there is a non-zero probability of successful message transmission.

5.3.1 Structure Specification of An Alternate-Bit Protocol

To guarantee a message sent from one end to be received finally at the other end, we should take advantage of the property, "non-zero probability of message transmission," of the unreliable medium. The idea is that the Sender keep on sending the same message unboundedly until it gets back an acknowledgement from the Receiver; and the Receiver acknowledges all messages received. To avoid duplication of messages, a serial (integer) number, as a unique id, is attached to each message sent by the Sender and the Receiver accepts messages only if their serial numbers have never appeared before. To avoid reordering messages, we sequentialize the sending of the message

by requiring that the Sender cannot send a second message until the previous one has been acknowledged.

The key ideas can be specified formally in EBS as follows:

```
(* Alternate-Bit Protocol *)

Sender:
     (* Guaranteed message transmission: keep
        on sending the same message
        unboundedly until get back an acknowl-
        edgement.  *)
          ∀ ip ← IP
            (∤ ds← DS ip=> ds) ^
            ((∤ ar← AR ar.msgno= ord(ip)) v
              (∀ d1← DS ip=> d1
                 #> ∤ d2← DS ip=> d2^d1->d2));

     (*Sequence Control:  do not send a new
       message until all previous ones are
       acknowledged.  *)
         ∀ ip← IP
           (∀ k← N
              k> ord(ip)
              #> ∤ ar ← AR ar.msgno =  k ^
                           ar ->ip);

     (* Contents of messages:  send out a
        message together with a serial number
        as a unique id.
        *)
          ∀ ip ← IP, ds ← DS
            ip => ds #> ip.msg = ds.data ^
                        ds.msgno = ord (ip);

Receiver
     (* Send acknowledgement for every mes-
        sage received back to the Sender.  *)
          ∀ dr← DR ∤ as← AS
            ^r=> as;

     (*send back the serial number as an
       acknowledgement of receipt.  *)
          ∀ dr ← DR, as ← AS
            DR => as #> as.msgno =
              dr.msgno;

     (* Accept those messages that never
        come before.  *)
          ∀ dr ← DR,
            ( ∤ op← OP dr => op)
            #> ~( ∤ dr'← DR
                  dr'-> dr ^
                  dr'.msgno= dr.msgno);
```

5.3.2 Verification of an Alternate-Bit Protocol

We want to prove that the structure specification of this Alternate-Bit Protocol meets its behavior specification. Since the DM (Data Transmission Medium) is an unreliable one, the SS (Send Station) has to send the messages unboundedly to guarantee that at least one will reach the RS Receive Station) finally. However, since the AM (Acknowledgement Transmission Medium) is also an unreliable one; it is possible that the acknowledgement may be lost, accordingly. Fortunately, it can be proved that if the SS sends the same messages unboundedly, though DM is unreliable, unbounded messages will arrive at RS. Since RS acknowledges all messages received, it is guaranteed that at least one acknowledgement will arrive at SS.

Theorem 2. If the underlying communication medium has a non-zero probability of message transmission, then if an unbounded number of messages having the same contents are sent from A, then not only one but an unbounded number of messages will arrive at B.

Proof By mathematical induction: Since unbounded number of messages having the same contents are sent from IP, at least one of them, say x, will reach OP. Since the number of messages after x is again unbounded, at least one of them will arrive OP. The same process goes on and on.

Theorem 3. The Alternate-Bit Protocol makes an unreliable system behave as a reliable one.

Proof Based on Theorem 2, the no loss property is easy to prove. Other properties can be proved one by one in a way similar to the proofs in the tandem network.

See [CHE82] for details of the formal specification and the verification of the Alternate-Bit Protocol.

6. Comparisons with Other Current Approaches

6.1 Temporal Logic Approaches

Temporal logic, first introduced by Pnuli in as an adaption of a classical logic suitable for defining the semantics of computer programs, is used in specifying and verifying concurrent systems [OWI80].

Several properties of concurrent systems can be stated using two temporal operations: □ (henceforth) and ◇ (eventually). However, global invariants that should be true throughout the computation, rather than merely input/output relations, are stated as the behavior specification of a distributed system. Though invariants facilitate implementation verification, they are difficult to specify, understand and are less intuitive than input-output relations from the user's viewpoint, as the behavior specification in EBS.

6.2 Trace Approaches

The notion of traces is used in the specifications and verifications of networks of processes by Misra & Chandy [MIS81], and Zhoa Hoare [ZHO81]. There are several deficiencies in the trace approach. First, since notations for sequences are used exclusively, trace specifications are awkward in expressing properties

whose data structures are not well-defined sequences. Typical examples are those properties of unreliable transmission systems that may lose, duplicate and reorder messages. Second, the "liveness" properties such as eventual deadlock, or eventual termination, etc., are in general not specified and verified using the trace notion directly.

In comparison, events in EBS are only partially ordered; no assumption of the existence of a global clock is made. The concept of events is more elementary than that of traces (sequences of events); consequently, some properties that can be specified in EBS easily can only be expressed in traces with difficulty. The "liveness" properties can be specified directly by the enabling relation => in EBS.

7. Conclusions

In summary, both the behavior and structural specifications based on event model are (1) formal: using partial ordering relations and first order predicate calculus; (2) minimal: orthogonal properties are specified separately; (3) extensible: new requirements can be added without changing the original specification; and (4) complete: most interesting properties in distributed systems can be specified.

The correctness of a design can be proved before implementation by checking the consistency between the behavior specification and structure specification of a system. Important properties of concurrent systems including both "safety" properties and "liveness" properties can be specified and verified.

Moreover, since the specification technique defines the orthogonal properties of a system separately, each of them can be verified independently. Thus, the proof technique avoids the exponential state-explosion problem found in state-machine specification techniques.

In addition to having the most desirable features of a specification technique, EBS represents time concept by a partial ordering relation of events and represents concurrency by the lacking of ordering between events. This makes EBS more accurate a model for distributed systems, which are inherently concurrent, asynchronous, and nondeterministic.

8. Acknowledgements

The authors would like to express their special appreciation for Prof. Tang Chih-Sung, whose comments concerning both the first order logic and distributed systems have been extremely helpful. Thanks, also, to Prof. Harlan Mills, John Gannon, Virgil Gligor and Pamela Zave for their guidance and comments on the early draft of this paper. This work was supported in part by the Air Force Office of Scientific Research, Contract AFOSR-F49620-80-C0001 to the University of Maryland.

9. References

1. [BAR69] Bartlett, K.A. et al. "Note on Reliable Full Duplex Transmission on Half Duplex Links," CACM 12(5): 260-261, May 1969
2. [CHE81a] Chen, B. and Yeh, R. T. "Event Based Behavior Specification of Distributed Systems," Proceedings of IEEE Symposium on 'Reliability in Distributed Software and Database Systems', July, 1981, Pittsburgh, Pennsylvania
3. [CHE81b] Chen, B. and Yeh, R.T. "Behavior Specifications of Distributed Systems," submitted to IEEE Transactions on Software Engineering
4. [CHE82] Chen, B. "Event Based Structure Specification and Verification of Distributed Systems," Ph.D. Dissertation, Computer Science Dept., University of Maryland, 1982
5. [GRE77] Greif, I. "A Language for Formal Problem Specification," CACM 20(12): 931-935, Dec. 1977
6. [HAI80] Hailpern, B. and Owicki, S. "Verifying Network Protocols Using Temperal Logic" in Trends and Applications 1980: Computer Network Protocols, IEEE Computer Society, May 1980
7. [HEW77b] Hewitt, C. and Baker, H.J. "Actors and Continuation Functionals" MIT/LCS/TR-194, 1977
8. [LAM78] Lamport, L. "Time, Clocks, and the Ordering of Events in a Distributed System," CACM 21(7) : 558-565, July 1978
9. [LAV78] Laventhal, M.S. "Synthesis of Synchronization Code for Data Abstractions" MIT/LCS/TR-203 Ph.D. Dissertation June 1978
10. [LIS77] Liskov, B.H. and Zills, S. "An Introduction to Formal Specifications and Data Abstractions" in Current Trends in Programming Methodology, Vol(1): 1-33, Yeh, R.T., Editor, Prentice Hall 1977
11. [MIS81] Misra, J. and Chandy, K.M. "Proofs of Networks of Processes" IEEE Transactions on Software Engineering SE 7(4) : 417-426, July 1981
12. [OWI80] Owicki, S. and Lamport,L. "Proving Liveness Properties of Concurrent Programs" Stanford University, Working Draft, Oct. 1980
13. [STE76] Stenning, N.V. "A Data Transfer Protocol Computer Networks 1(2) : 99-110, Sept. 1976
14. [YEH80] Yeh,R.T. and Zave,P. "Specifying Software Requirements,"Proc. of IEEE, Oct. 1980
15. [ZHO81] Zhoa, C.C. and Hoare,C.A.R. "Partial Correctness of Communicating Sequential Processes" Proc. of IEEE, 2nd International Conference on Distributed Computing Systems, Paris, France, April 1981

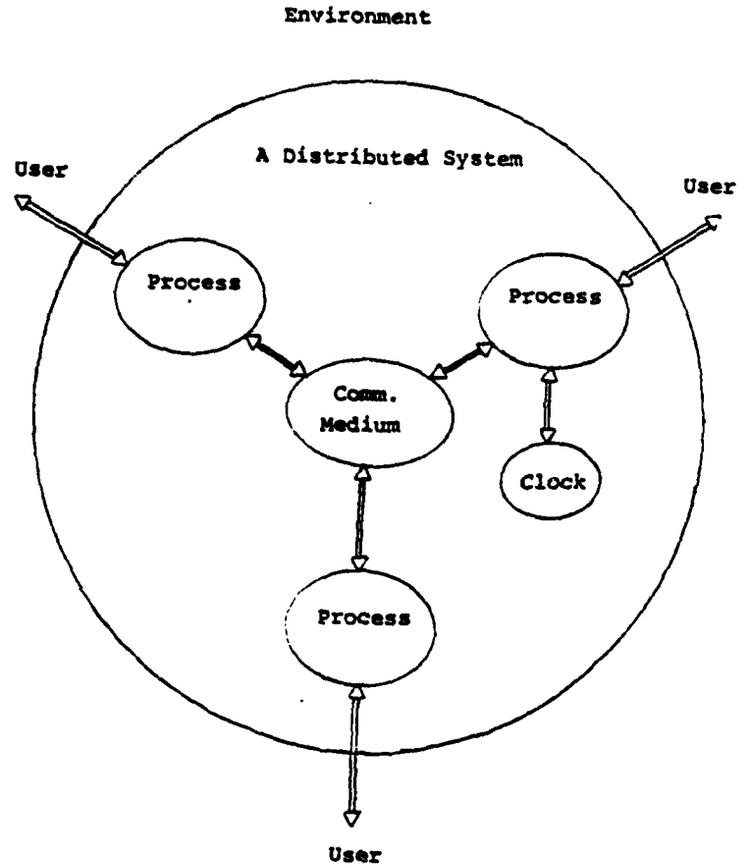


Figure 1. A Distributed System:
A Designer's View

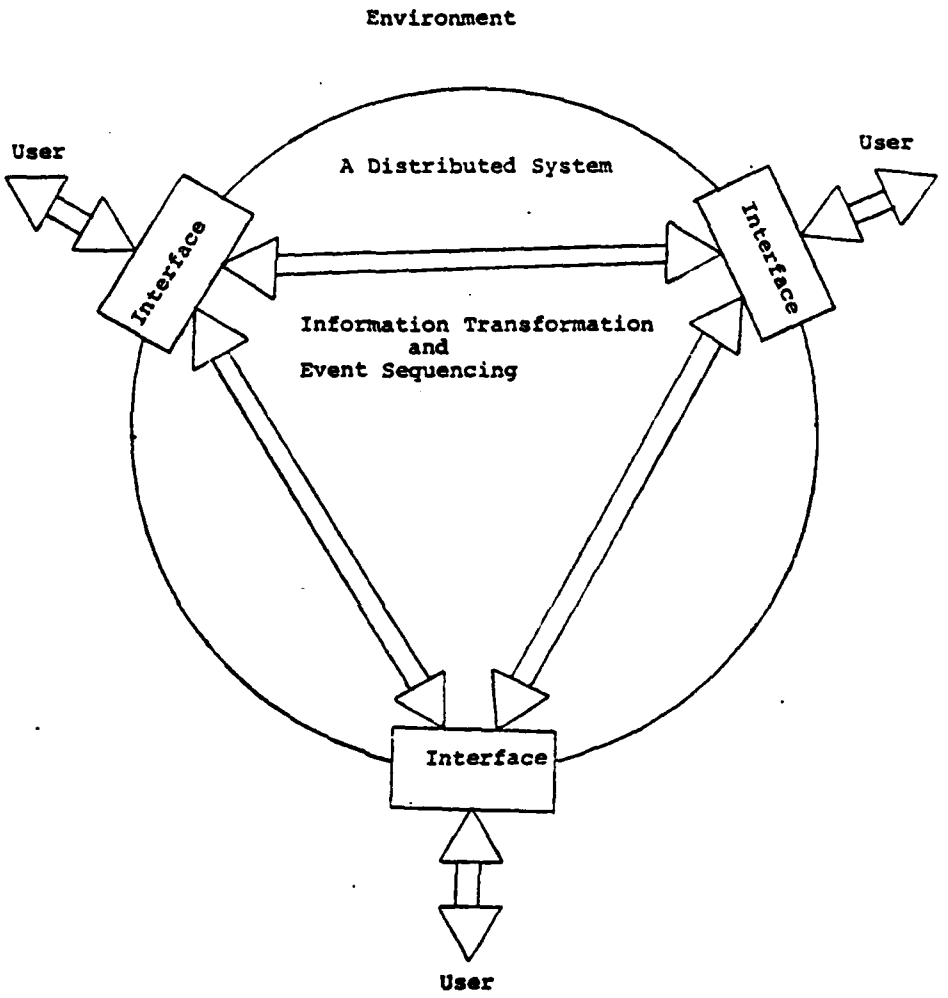


Figure 2. A Distributed System:
A User's or A System
Analysist's View

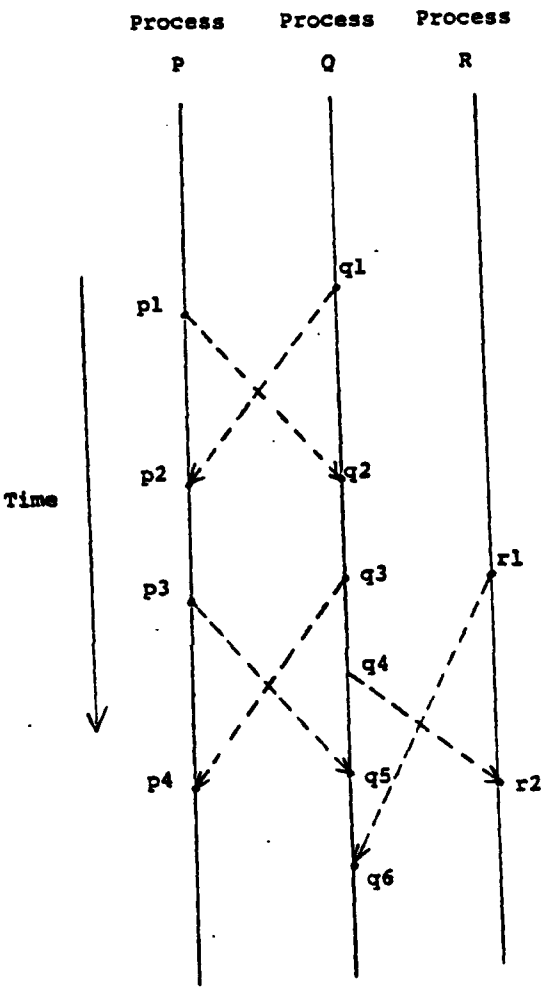


Figure 3. Preceding Relation Between Events in Distributed Systems

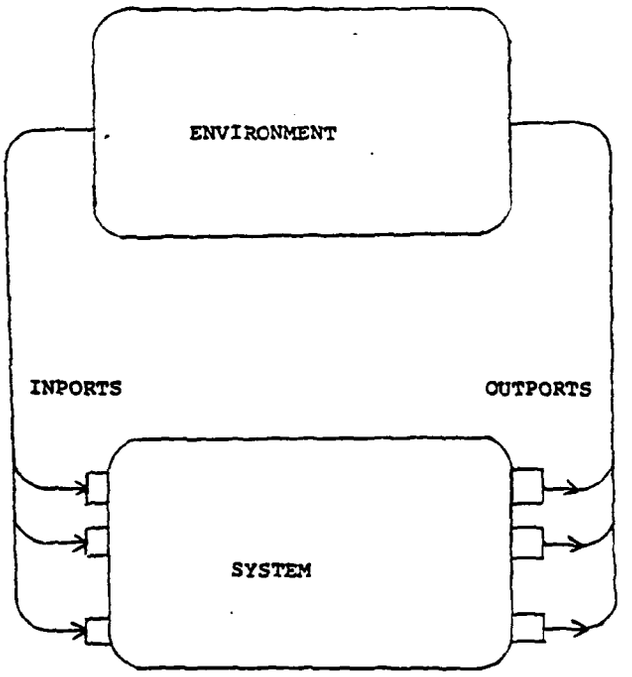


Figure 4. System, Environment
and Their Interfaces

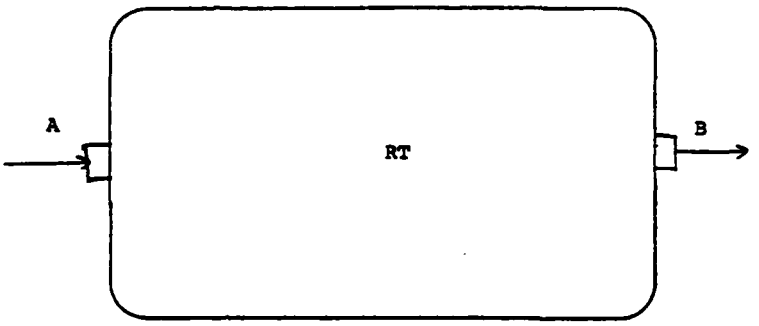


Figure 5. A Reliable Transmission System

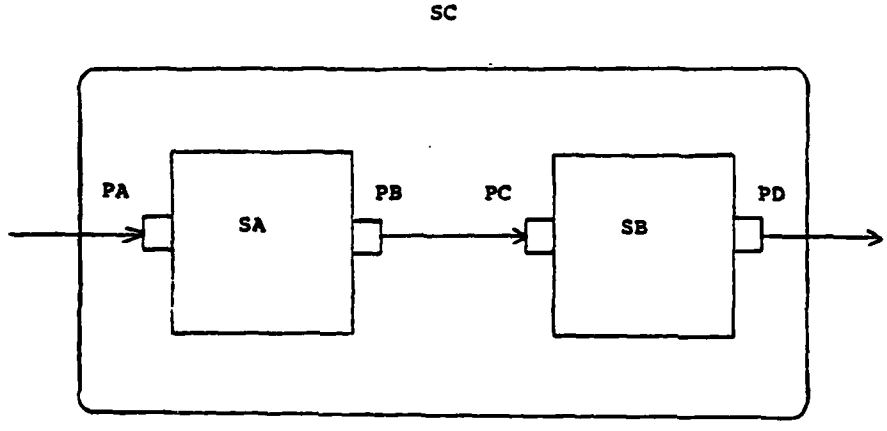


Figure 6. A Tandem Network

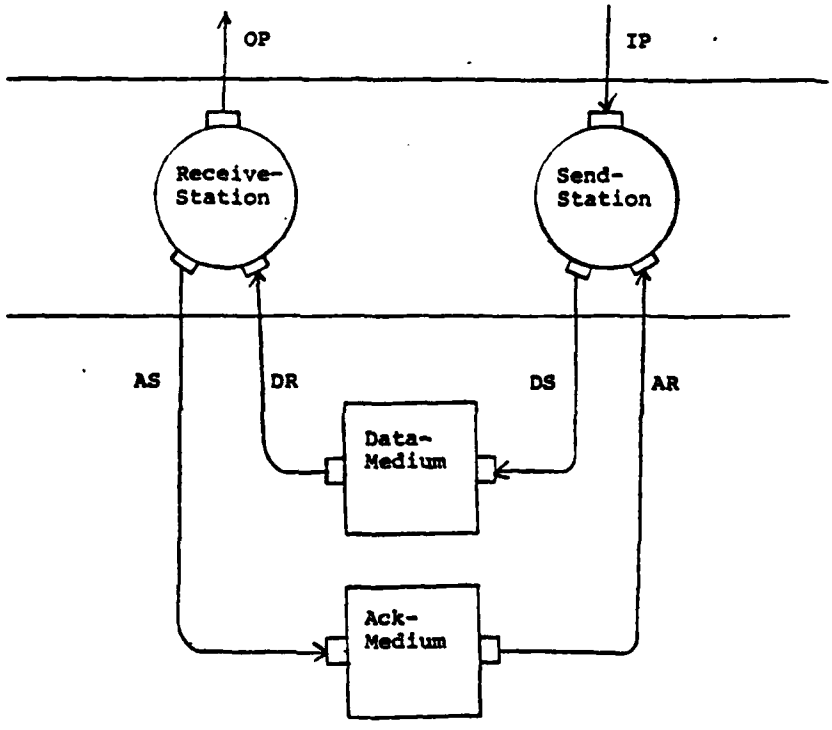
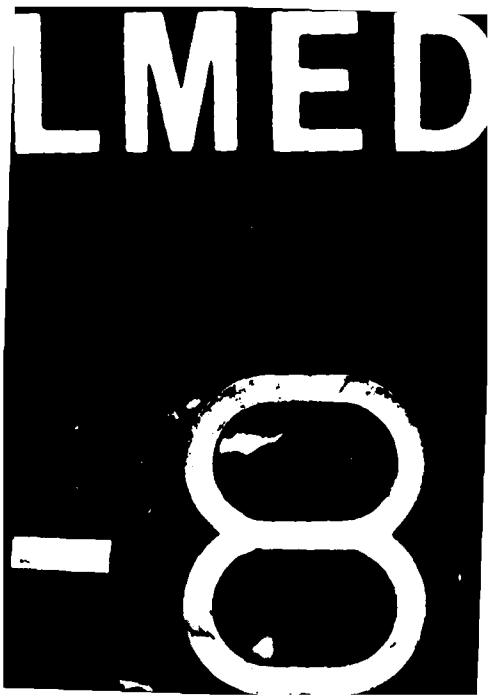


Figure 7. An Implementation Structure
Of The Alternate-Bit Protocol


LMED
8